Euso Simulation and Analysis Framework

User Guide

Feb 26, 2004 - Version 1.0
Doc. Ref. EUSO-SDA-REP-014-1

M. Pallavicini [1] and **A. Thea** [2]

Istituto Nazionale di Fisica Nucleare & Università di Genova
Italy

Abstract

This is the ESAF User Guide

[1]e-mail: Marco.Pallavicini@ge.infn.it
[2]e-mail: Alessandro.Thea@ge.infn.it

Contents

1 Introduction

ESAF stands for Euso Simulation and Analysis Framework. It is a software framework developed for the Extreme Universe Space Observatory, EUSO.

It is an integrated software designed to handle the event simulation chain (shower development simulation, light production due to fluorescence and Cerenkov, atmospheric effects, light transport to the Euso detector and the response of the Euso detector itself) and the reconstruction and analysis of both simulated events and of the real events.

This note is a simple User Guide, therefore very little space is devoted to the description of the internal structure of ESAF and of the code. The reader should look at the ESAF document in the bibliography.

The topic addressed in this document are basically the following:

- Requirements
- Getting the source code
- Compiling, linking and updating
- Running Simu
- Output Root file structure
- The reconstruction code

The ESAF code is written in C++ and Fortran and is based on the ROOT package [2]. Although the code is written in a highly portable way, because of lack of man power we support the Linux platform only.

The compiler is the standard gcc version 3.2 or higher [3]. Several Linux distributions have been used in the recent years without major problems.

Even if we do NOT give any support, we encourage the user to try to port the code to different UNIX platforms if needed, because it should not create major problems.

1.1 Requirements

gcc version 3.2 or higher must be installed on your system. Type `gcc -v` to get the version. If you get an error here, you don't have `gcc` at all!

ROOT must be installed and configured; the `ROOTSYS` environment variable must be properly set and must point to a recent version of ROOT (3.10.2 or higher). Development is carried on using the last **pro** version available (differences among **new**, **pro** and **old** are explained on ROOT website) therefore we strongly suggest keep the ROOT version up to date. If you do not know how to set the `ROOTSYS` variable, ask your local system manager.

g77 The Fortran compiler sometimes is not included in the Linux distribution. Make sure that it is installed on your machine and eventually contact your computer administrator to get it.

zlib We use compressed ASCII files and we need the zlib library [4]. This is a very standard library available for any UNIX platform. Be sure that you have it.

1.2 Getting the code

The ESAF code is available through **CVS** (Concurrent Version System) [5] from the CVS server at the Lyon in2p3 Computing Center. The code is available in "read-only" mode for normal users and in read/write mode for developers.

In the normal read-only mode, the user should configure his ssh directory first. Inside your directory `.ssh`, create an ASCII file named `config` with the following lines:

```
Host cvs.in2p3.fr
     Port 2222
     User euso
     PasswordAuthentication yes
     RSAAuthentication no
     PubkeyAuthentication no
     ForwardX11 no
     ForwardAgent no
```

Then the user should set the following UNIX environment variables (see CVS manual for details):

```
[user]# export CVSROOT=euso@cvs.in2p3.fr:/cvs/euso/
[user]# export CVS_RSH=ssh
```

After these settings are done you are ready to get the code with the standard CVS command:

```
[user]# cvs co esaf
```

The developer that needs write acces to the CVS repository must get in contact with the ESAF group and get an account at the Lyon CC. The complete documentation about Lyon CVS accounts are available on the Lyon CC webpage [6].

1.3 Compiling and Linking

Once you have the ESAF source code, compiling is a very simple task. Just enter into the ESAF directory that CVS has created in the place where you typed `cvs co esaf` and then type `make`.

The compilation and linking may take several minutes, depending on the machine you are using. If everything went fine, at the end you should have 2 executable files in the `bin/i686` directory, named `Simu` and `Reco`.

Compilation and or Linking may fail for several reasons; if so, and you didn't modified the code, make sure that all requirements are satisfied.

1.4 Updating the code

ESAF is continuously updated (developing and debugging); in most cases changes are small and don't affect ESAF's behaviour deeply. On the contrary sometimes are so significant to require a total rebuild of the code and may and affect also the output files so that they are not compatible between different versions of the program. Therefore we advise to keep the code up-to-date. To do this just go in `esaf` directory and type at the system prompt

```
[user]# cvs up -d
```

This updates files that are changed, deleted or created; the optiona flag `-d` is needed when new directory has been added to the repository. This doesn't occur often, so basically you can just type `cvs up`.

It is also possible to update each directory or file indipendently running `cvs up` from the directory itself (as long as in that directory there is a subdirectory `CVS`).

There is an ESAF CVS mailing list that keeps the users informed whenever new code is committed to the repository and that sends the list of the modified files. If you are interested to receive these mails, you are invited to contact the authors.

1.5 Building options

In the main ESAF directory the `Rules` file contains the compilation options.

The most important option for the user is `ESAFTMP`; this variable points to the directory where `make` saves the temporary object files for creation of libraries and binaries. Default directory is:

```
ESAFTMP = /tmp
```

In some cases is advisable to change the default directory (editing the file `Rules`) because the directory `/tmp`, present in all systems, is generally emptied when you reboot the machine.

In `ESAFTMP` directory contains `ESAFTMP.username/architeture/` in which are other subdirectories, one for each library of ESAF.

1.6 Cleaning and rebuilding

Sometimes, after a major change or when an obsolete file is removed from repository `make` could fail to rebuild ESAF. This is due to old temporary files in `ESAFDIR` that conflicts with the new code. To get rid of them two ways exist.

In most cases `make` is able to clean `ESAFDIR` by itself. Just go in `esaf/` directory and type

```
[user]# make clean
[user]# make
```

If also `make clean` fails or the compilation is still broken, `ESAFDIR` must be deleted by hand.

```
[user]# cd [ESAFTMP]
[user]# rm -rf
[user]# cd [ESAFDIR]
[user]# make
```

As last chance, if you still have problems, you can delete the entire `esaf/` and the `ESAFTMP` directories and reinstall ESAF.

The other options in the file `Rules` is `DEBUG`, for compiling with (`DEBUG = 1`) or without (`DEBUG = 0`) debugger support.

2 Running Simu

If you successfully compiled and linked ESAF, you have two binary files in your `esaf/bin/i686` directory.

Before running, check that the environment variable `LD_LIBRARY_PATH` includes the esaf/lib/i686 directory. If this is not the case, type the command:

```
[user]# export ESAFDIR = /home/user/esaf;
[user]# export LD_LIBRARY_PATH=$(LD_LIBRARY_PATH):$ESAFDIR/lib/i686
```

This variable tells the operating system where the dynamic shared libraries are located. Depending of your system setup, you might have to add the `$ROOTSYS/lib` directory to the same path. You can check whether it is already defined or not by typing `echo $ROOTSYS`.

The first binary is Simu. It is the simulation program with the graphical user interface.

The simplest way to run Simu is the following:

```
[user]# bin/i686/Simu -b --events=nev
```

The `-b` option means that you are running in batch mode, without interactive windows. If you remove this option, a Graphic User Interface starts. This GUI is right now a very incomplete demo and should not be used at this stage of development.

The `--events` option specifies how many events simulate. Default is 1.

All other parameters (a very large number of parameters!) are hidden in the configuration files in the directory `esaf/config`.

To change the behaviour of the program right now the best way is to edit these files. The meaning of each variable is described into the files themselves. In the next section we give just the list of these files with their general meaning. Please refer to the code to have more details.

2.1 The configuration files

Most of the ESAF behaviour, both running Simu and Reco, is controlled by the value of a set of variables written into several configuration files.

All configuration files are stored in the `esaf/config` directory tree, whose structure is described in section C.

A Standard directory is also foreseen. This directory will contain a set of standard, well identified configurations. The idea behind this is that no user will normally have to change the config files; it will use a standard configuration (selectable from GUI or with inline command) and will change a very small number of parameters (again from GUI or inline command). This will be ready soon.

2.2 The Root File Format

The ESAF output is twofold: a ROOTfile with the descripton of the events and the detector configuration and a gzipped ASCII ”telemetry” file.

The Root file contains two object of type `TTree`. The former, `etree`, is basically a collection of `EEvent` objects, each of them containing all information concerning one event. The latter, `runpars`, holds a set of maps with the parameters of the detector (`ERunParameters`) during the events simulation.

For a deeper understanding of the TTree object or any other relevant concept related to ROOT the reader should read the ROOT documentation.

The `EEvent` object is made of a set of list of sub-objects (`TClonesArrays` of objects) and some single sub-objects.

The `EEvent` elements are the following:

- `fHeader` This is a single object of type EHeader. It contains general infos about the event. Right now only:
  - `fHeader.fNum` Event number. Progressive starting from 0.
  - `fHeader.fRun` Run number. Fixed and not relevant for MonteCarlo right now. It will be important when a complete data base handling will be included.
- `fTruth` This is a single object of type `ETruth`. It contains general infos about the Monte Carlo truth.
  - `fTruth.fTrueEnergy` Primary EECR energy in eV
  - `fTruth.fTrueTheta` Incidence angle (rad) from normal to earth in shower ref. system
  - `fTruth.fTruePhi` Azimuth ($\phi = 0$ corresponds Y=0)
  - `fTruth.fTrueParticleName[20]` Particle name
  - `fTruth.fTrueParticleCode` Same as name with code instead of strings
  - `fTruth.fTrueInitPos[3]` First interaction point (3D coord, Km)
  - `fTruth.fTrueX1` Interaction depth in g/cm^2
  - `fTruth.fTrueEarthImpact[3]` Impact of shower on earth (clouds ignored)
  - `fTruth.fTrueEarthAge` Age of the shower at impact
  - `fTruth.fTrueShowerMaxPos[3]` Shower max position (3D coord, Km)
  - `fTruth.fTrueShowerXMax` Shower max depth in g/cm^2
- `fShower` This is a single object of type `EShower`. It contains general infos about the Monte Carlo truth of the Shower only. More detailed informations. Not filled yet.
- `fDetector` This is a single object of type EDetector. Very general infos about detector response:

  - fDetector.fNumGtu Total number of GTUs in which there was activity
  - fDetector.fGtuStart First GTU with data
  - fDetector.fGtuEnd Last GTU in data
  - fDetector.fNumCell Number of macrocells with data
  - fDetector.fTimeFirstGtu Time (ns) corresponding to first edge first gtu
  - fDetector.fTimeLastGtu Time (ns) corresponding to second edge last gtu
  - fDetector.fGtuLength Time duration (ns) of one GTU

- fNumPhotons Single integer variable. Number of photons in fPhotons array
- fNumFee Single integer variable. Number of EFee objects in fFee
- fNumAFee Single integer variable. Number of EAFee objects in fAFee
- fNumCells Single integer variable. Number of EMacrocell objects in fMacrocell.
- fNumCellHits Single integer variable. Number of EMacroCellHit objects in fData.
- fFirstTime Single float variable. Time of the first photon entering pupil (ns)
- fLastTime Single float variable. Time of the last photon entering pupil (ns)
- fPhotons This is a TClonesArray of objects EPhoton. Each EPhoton contains the whole history of any physical photon that has entered the detector and has been traced inside Euso. In the following, nn is an index from 0 to fNumPhotons-1.
  - fPhotons[nn].fType Photon type (Cerenkov or fluorescence or nightglow)
  - fPhotons[nn].fState True if photon absorbed in atmosphere
  - fPhotons[nn].fShowerPos[3] Position in the Shower
  - fPhotons[nn].fTheta Photon direction at pupil
  - fPhotons[nn].fPhi " " " "
  - fPhotons[nn].fLambda Photon wave length
  - fPhotons[nn].fTime Time at pupil
  - fPhotons[nn].fHitOnIFS True if photon crosses the Ideal Focal Surface
  - fPhotons[nn].fMadeSignal True if signal in the pmt
  - fPhotons[nn].fMadeCount True if signal was counted in chip
  - fPhotons[nn].fMadeFastOR True if was counted in macrocell

- fPhotons[nn].fHistory Code of last position of the photon
- fPhotons[nn].fFate Return flag of the module in which photon ends
- fPhotons[nn].fMacroCell Macrocell number hit
- fPhotons[nn].fPmt Pmt number
- fPhotons[nn].fPmtCh Pmt channel
- fPhotons[nn].fFe Fe chip
- fPhotons[nn].fFeCh Fe channel
- fPhotons[nn].fGtu Gtu number this photon belongs to
- fPhotons[nn].fPixelUid Unique pixel id
- fPhotons[nn].fSignalId Identifier of the PmtSignal object
- fPhotons[nn].fXCell X coordinate in macrocell (column)
- fPhotons[nn].fYCell Y coordinate in macrocell (row)
- fPhotons[nn].fPos[3] Last known position of the photon
- fPhotons[nn].fIdealFocalPos[3] Euso coordinate on the ideal focal surface
- fPhotons[nn].fCharge Pmt charge associated with this photon.
- fPhotons[nn].fIPeak Peak current at input of front end chip.
- fPhotons[nn].fSignalTime Time when pmt signal occurs (ns)

- fData This is a TClonesArray of objects EMacrocellHit. Each EMacrocellHit contains the relevant data for single pixel seen by a Macrocell above threshold. In the following, nn is an index from 0 to fNumMacrocellHits-1.
  - fData[nn].fCellId MacroCell identifier
  - fData[nn].fRow X coordinate internal to macrocell (integer)
  - fData[nn].fCol Y coordinate internal to macrocell (integer)
  - fData[nn].fGtu Gtu number
  - fData[nn].fTheta Theta angle seen by the pixel in field of view
  - fData[nn].fPhi Phi angle seen by the pixel in the field of view
  - fData[nn].fTime Time relative to first triggering GTU

- fFee This is a TClonesArray of objects EFee. Each EFee contains the relevant data for single pixel seen by the front end electronics. In the following, nn is an index from 0 to fNumFee-1.

- `fFee[nn].fMCId` MacroCell identifier
- `fFee[nn].fGtuId` Gtu number
- `fFee[nn].fFEId` Front end chip (integer)
- `fFee[nn].fChanId` Unique chanel id (integer)
- `fFee[nn].fNumHits` Number of hits counted
- `fFee[nn].fTh` Theta angle seen by the pixel in field of view
- `fFee[nn].fPh` Phi angle seen by the pixel in the field of view
- `fFee[nn].fHasTriggered` flag if there was trigger or not
- `fFee[nn].fCharge` Collected charge in this pixel

- `fAFee` This is a `TClonesArray` of objects `EAFee`. Each `EAFee` contains the relevant data for single pixel seen by the analog front end electronics. In the following, `nn` is an index from 0 to `fNumAFee-1`.
  - `fFee[nn].fMCId` MacroCell identifier
  - `fFee[nn].fGtuId` Gtu number
  - `fFee[nn].fFEId` Front end chip (integer)
  - `fFee[nn].fDyCharge` Dynode charge in this gtu
  - `fFee[nn].fCherTrigg` flag for cerenkov trigger
- `fMacrocell` This is a `TClonesArray` of objects `EMacrocell`. Each `EMacrocell` contains the relevant data for macrocell that has detected at least one photon. In the following, `nn` is an index from 0 to `fNumCells-1`.
  - `fMacrocells[nn].fMCId` Macrocell id
  - `fMacrocells[nn].fNumChips` Number of front end chips giving signal
  - `fMacrocells[nn].fNumPixels` Number of pixels with at least one photon
  - `fMacrocells[nn].fNumCounts` Number of counts detected by chips (not necessarily mcell)
  - `fMacrocells[nn].fNumFastOrs` Number of fast or counts detected by macrocell
  - `fMacrocells[nn].fHasTriggered` Trigger condition occured
  - `fMacrocells[nn].fGtuTrigger` Gtu number when trigger occurred
  - `fMacrocells[nn].fTriggerWord` Trigger engine identifier word (bitfield)

3 The Reconstruction code

3.1 Overview

In this section we describe the structure of the Reco code. The logic here is slightly different from the previous section, because even normal user must have a good understanding of the Reconstruction framework in order to use it efficiently. Therefore, some more insight of the internal structure of the framework is given, even if Rule number 0 of the software developer should always be remembered: the only document that is always up to date is the code!

The proposed scheme for the organization of the ESAF reconstruction module is sketched in figure 2. Input data comes either from the simulation module (Root file) or from the real data stream (pre-processed telemetry)[3].

From this figure, the following basic entities/structures can be identified:

- Input module

  This modules handles the reading of the events for reconstruction. Foreseen are the reading of a Root simulation file (already supported) and the reading of events from the real data stream (pre-processed telemetry).

- Event container

  This is the container structure for the input event which we are going to reconstruct. The event container will allow the access the objects holding: event header information, trigger information and readout information at macrocell and pixel level. In principle the available information should correspond exactly to the one available in real data, although test modes in which the "Monte Carlo truth" is available are also foreseen.

- Reconstruction framework

  This is the main structure, which actually builds the chain of modules which will reconstruct the event. Event reconstruction is divided into different tasks (e.g. direction reconstruction, energy reconstruction, ...) and for each task different possible modules may be available. The use of a structure allows to easily replace or exclude a given processing module. In this way different algorithms or algorithm combinations can be compared and tests can be performed.

- Modules

  This is the set of processing modules, possibly several alternative modules for each specific task, that can actually be picked by the user and included in the reconstruction chain build by the framework.

[3] Right now, only data input from simulation ROOTfile is available

- Configuration files

  Configuration data includes the name and location of the input file and the definition of the processing modules to load.

  The configuration definition in Reco is rather simple for the user point of view but corresponds to a relatively complex implementation (see below). It allows to change the base directory for all configuration files, thus allowing to define and use different full configurations stored in different places.

  The reconstruction uses the files stored in `config/Reco` directory.

- Access to databases

  Access to databases will be a major issue in Reco. The reconstruction procedure will naturally require the access to a run conditions database, as well as to the detector calibration database. Furthermore, access to atmospheric data (both collected by the EUSO atmospheric sounding devices or originating from external sources and databases) will have to be accessed.

  This part of the code is, at this early stage, not yet implemented. See below (future developments) for more information.

This structure is shown in the interactions diagram in figure D.

3.2 Class structure

The basic class diagram of Reco as its present stage is schematically shown in figure D.

The different basic "groups" in the above classification are shown in different colours. The scheme is described in the sections below, where some design and implementation aspects are presented.

3.2.1 Input

There is an abstract interface `InputModule` from which are built the different concrete input classes. The existing concrete classes are `RootInputModule` (for simulation file reading) and `TestInputModule` (for test purposes only, not to be used by normal users).

3.2.2 Event

The main class is the event container, `RecoEvent`. The event header is stored separately in a RecoEventHeader class. For each `RecoEvent` a `RecoEventHeader` is required. Event information is kept in the objects `RecoCellHit`, with macrocell level information (this is the class defining a hit), `RecoPixelData`, a generalisation of the previous one containing pixel level information. Furthermore, `RecoCellInfo` contains trigger and macrocell level information and `RecoAnalogData` holds analog readout information and `RecoPhotoElectronData` contains informations about the photoelectrons for test and debugging.

3.2.3 Framework

The *framework* is responsible for creating a *factory* which will generate *modules* according to pre-defined models. Different module types can be selected/unselected. One and only one input module should exist.

The central class is the `RecoFramework` class, which will create a *factory*, `ModuleFactory`, able to generate modules according to a given model. `RecoModule` is the abstract interface from which are built the different concrete classes defining the different types of modules. `TestRecoModule` is an example concrete implementation. `RecoSequence` is meant to define a special type of composite module.

3.2.4 Modules

A set of modules will be available for the different reconstruction tasks. As an example, a basic clustering algorithm is implemented in the class `BaseClusteringModule`, which, just like `TestRecoModule`, inherits from the abstract interface `RecoModule`. This section of the code is currently being implemented, the diagram corresponds to the status on January 23^{rd}.

3.2.5 Configuration and utilities

In this category are included different software tools of interest for both the simulation and the reconstruction parts of ESAF. This includes code for the following purposes: configuration handling, object persistency (ROOT), graphic user interface, basic data and mathematical procedures (units, constants, random number handling, time/orbit info, ISS time/orbit description...), atmosphere description. As seen in the diagram, at present configuration handling is already implemented and can serve here as an example: `EusoConfigurable` is the abstract interface that should be used by all classes that need to access configuration data, in order to create `Config`, a singleton object in which the relevant information is stored. All the classes that need to access configuration data should inherit from `EusoConfigurable` (and call in their definition the macro `EusoConfigurable(type,name)`. The method `Conf()` returns a pointer to a `ConfigFileParser` object that contains all the parameter values (identifies the file containing the configuration parameters for the object considered, parses the file, and stores the parname=parvalue pairs in maps). The `ConfigFileParser` objects are created by a factory, `Config`, This object is a singleton.

3.3 Time sequence

This section presents a rough and simple-minded temporal sequence view of the control flow and object interaction during and Reco test run. The purpose is to clarify the ideas and provide the reader with a simple picture of how it works. A schematic representation is given in figure D. In the description below, the pure object lifecycle view is here mixed with a more code-oriented description of the program flow (usually given in parentheses).

- Building the Framework:

  1. A `RecoFramework` object is created (`RecoFramework` constructor called in `reco-main.cc`):

```
RecoFramework theFrameWork;
```

  2. get name of the file with module list and read module list (within `RecoFramework` constructor);

```
string sName = Conf()->getStr("RecoFramework.ModuleFile");
sName = "./config/Reco/"+sName;
```

  3. create factory object `ModuleFactory` and build module (within `RecoFramework` constructor, calling `ModuleFactory` constructor with module list as arguments. In there, modules are built using the `MakeModule()`, `MakeInputModule()` methods of the `ModuleFactory`).

```
// build factory
ModuleFactory factory( sName );

// build modules
if ( identifier == "InputModule" ) {
    MakeInputModule(name);
}
else if ( identifier == "Module" ) {
    MakeModule( name );
}
else if ( identifier == "Sequence" ) {
    MakeSequence( name );
}
else {
    throw runtime_error("Syntax error in file"
```

```
                                            +sName+" line "+dummy);
}
```

- Executing the module chain:

  1. Call `Execute()` method of `RecoFramework` (back to main, perform some information dump and then execute framework);

```
theFrameWork.Dump();
theFrameWork.ParseCommandLine(argc,argv);
try {
    theFrameWork.Execute();
}
catch( exception &e ) {
    cerr << "RECO Error: " << e.what() << endl;
cerr << "Euso Reco Program Exiting" << endl;
exit(1);
}
```

  2. initialize input module and the other modules (`Init()` methods of `TestInputModule` and `TestRecoModule` classes);

```
// init input module
fInputModule->Init();

// init all modules
for( it = fModules.begin(); it != fModules.end(); it++) {
    if ( ! it->second->Init() ) {
    cerr << "Module " << it->first << " failed\n";
    throw runtime_error("Init failed");
}
```

  3. get event into a `RecoEvent` object (a `RecoEvent` object is created and the `GetEvent`() method of `TestInputModule` is invoked);

```
// run
while ( RecoEvent *anEvent = fInputModule->GetEvent() ) {
```

  4. process the event through each module (the methods `PreProcess()`, `Process()` and `PostProcess()` of TestRecoModule are invoked);

```
for( it = fModules.begin(); it != fModules.end(); it++) {
    if ( ! it->second->PreProcess() )
    break;
if ( ! it->second->Process( anEvent ) )
    break;
if ( ! it->second->PostProcess() )
    break;
}
```

5. destroy the event (method `DestroyEvent()` of `TestInputModule`);

```
fInputModule->DestroyEvent();
```

6. done with all modules (`Done()` methods of input and other modules).

```
// end all modules
fInputModule->Done();
for( it = fModules.begin(); it != fModules.end(); it++) {
    it->second->Done();
}
```

Procedures 1 to 5 are obviously repeated for each event, and persistency is handled before the event is destroyed.

A Acknowledgements

We thank C. Espirito Santo for her help on the reconstruction sections and for several pictures. We also thank R. Pesce for several useful comments.

B References

References

[1] D. De Marco and M. Pallavicini, EUSO-SIM-ESAF-001-01, available on LiveLink

[2] The ROOT System http://root.cern.ch

[3] GNU Compiler Collection Homepage http://gcc.gnu.org

[4] Zlib library homepage http://www.gzip.org/zlib

[5] Concurrent Version System http://www.cvshome.org

[6] Service CVS au Centre de Calcul de l'IN2P3 http://cvs.in2p3.fr

C Configuration files references

In this section are described very briefly most of the variable of the `config` directory. Basically each file is associated to the class with the same name and format used for the variables is the following:

```
<ClassName>.<VariableName> = <Value>
```

Anyway there exists files like `Run.cfg` and `Euso.cfg` that holds general purpose variables. Due to a lack of space, in the tables `Filename` has been stripped.

Table 1: Variables stored in files of directory `config/General`

Variable Name	Units	Description
		`Euso.cfg`
`fRadius`	m	Euso radius
`fAltitude`	km	Euso altitude
`fEarthRadis`	km	Earth radius
		`Run.cfg`
`fTelemetryOutputFile`	-	Name of the telemetry file
`fRootOutputFile`	-	Name the root file
`fEnableRoot`	-	Enables rootfile output
`fRunNumber`	-	Run number
`fRunDate`	-	Date of the run

Table 2: Variables stored in files of directory `config/LightToEuso`

Variable Name	Units	Description
		`StandardLightToEuso.cfg`
`fGenerator`	-	Event generator choice
`fLightSource`	-	Source of light choice
`fRadiativeTransfer`	-	Radiative transporter choice
		`TestLightToEuso.cfg`
	TO BE DONE	
		`FileUnisimLightToEuso.cfg`
	TO BE DONE	
		`SlastLightToEuso.cfg`
`fEarthRadius`	km	Earth Radis
`fISSVectorX`	km	EUSO X coordinate from ground
`fISSVectorY`	km	EUSO Y coordinate from ground

Table 2: (. . . continued)

Variable Name	Units	Description
`fISSVectorZ`	km	EUSO Z coordinate from ground
`fFOV`	deg	EUSO Field of View
`fEntrancePupilDiameter`	m	EUSO Entrance Pupil Diameter
`fWaveRangeMin`	nm	Minimum wavelength
`fWaveRangeMax`	nm	Maximum wavelength
`fDoCherenkov`	-	Enable Cherenkov
`fDoFluorescence`	-	Enable fluorescence
`fAtmosphericType`	-	Atmosphere profile
`fAtmTemperature`	K	Atmosphere Temperature (valid only for US Standard)
`fAlbedo`	-	Earth Albedo
`fGTU`	μs	EUSO Gate Time Unit
`fAtmCurvature`	-	Atmosphere (*Curved* or *Planar*)
`fInteractionVectorX`	km	first point of shower (X coord.)
`fInteractionVectorY`	km	first point of shower (Y coord.)
`fInteractionVectorZ`	km	first point of shower (Z coord.)
`fThetaRangeMin`	deg	Θ_{min}
`fThetaRangeMax`	deg	Θ_{max}
`fEnergyRangeMin`	eV	Minimum energy to generate
`fEnergyRangeMax`	eV	Maximum energy to generate
`fRandomEnergy`	-	Energy randomizing algorithm
`fUhecrType`	-	UHECR type: – Atomic mass for nuclei – 1001 for ν
`fEnergyDistribution-Parametrization`	-	Parametrization for the energy distribution

Table 3: Variables stored in files of directory `config/RadiativeTransfer`

Variable Name	Units	Description
ALL TO BE DONE		
		`Ground.cfg`
		`Lowtran.UserModel.cfg`
		`Lowtran.cfg`
		`LowtranFactory.cfg`
		`RadiativeFactory.cfg`

Table 3: (…continued)

Variable Name	Units	Description
		`RadiativeTransfer.cfg`

Table 4: Variables stored in files of directory `config/Atmosphere`

Variable Name	Units	Description
	ALL TO BE DONE	
		`AtmosphereFactory.cfg`
		`LinsleyAtmosphere.cfg`
		`MSISE_00Atmosphere.cfg`
		`MSISE_00AtmosphereData.cfg`

Table 5: Variables stored in files of directory `config/Electronics`

Variable Name	Units	Description
		`FrontEndChip.cfg`
`fTimeResolution`	ns	Chip resolving time
`fGain`	-	PreAmp gain
`fThreshold`	μA	Current thresh.
`fCounterThreshold`	counts	Digital thresh.
`fGainSpread`	-	Spread in PreAmp gain
`fThreshSpread`	μA	Spread in curr. thr.
`fTriggerGroups`	-	Group logic code
		`R7600M64Photomultiplier.cfg` *and* `R8900M36Photomultiplier.cfg`
`fPmtQuantum`	-	Quantum efficiency
`fPmtGain`	-	Charge Gain
`fPmtGainsigma`	-	Spread in Charge Gain
`fPmtTimeWidth`	ns	Signal base width
`fPmtDoNightGlow`	bool	Add night glow if true
`fPmtNightGlowRate`	GHz	NG rate per pixel
`fPmtSide`	-	Number of channels
`fPmtSize`	mm	Physical size lateral
`fPmtDeadLateral`	mm	Dead space at border
`fPmtDeadInner`	mm	Dead internal space
		`ElectronicsFactory.cfg`
`fMacroCellType`	string	Type of macrocell

Table 5: (...continued)

Variable Name	Units	Description
fFrontEndType	string	Type of FE chip
fPmtType	string	Pmt Type
fPmtFile	string	obsolete
fFile	string	obsolete
fTelemetryType	string	output file format
fAfeeType	string	Select AFEE
fElementaryCellType	string	Type of macrocell

Table 6: Variables stored in files of directory `config/Optics`

Variable Name	Units	Description
ALL TO BE DONE		
		DetectorTransportManager.cfg
		ElecTestDetTransManager.cfg
		FakeOpticalAdaptor.cfg
		FastFocalPlane.cfg
		FileGenerator.cfg
		IdealOpticalAdaptor.cfg
		JIdealFocalSurface.cfg
		KIdealFocalSurface.cfg
		KOpticalSystem.cfg
		LensGenerator.cfg
		OpticsAnalyzer.cfg
		PipesOpticalAdaptor.cfg
		TestBaffle.cfg
		TestFocalPlane.cfg
		TestOpticalAdaptor.cfg
		TestOpticalSystem.cfg
		WallInteraction.cfg
		EusoMapping.cfg
		OpticsFactory.cfg

Table 7: Variables stored in files of directory `config/LightSource`

Variable Name	Units	Description
ALL TO BE DONE		

Table 7: (. . . continued)

Variable Name	Units	Description
		CrkvPhoton.cfg
		FileLightSource.cfg
		FluorescenceFactory.cfg
		LidarLightSource.cfg
		LightSource.cfg
		LightningLightSource.cfg
		MeteoritesLightSource.cfg
		TestLightSource.cfg

D Pictures

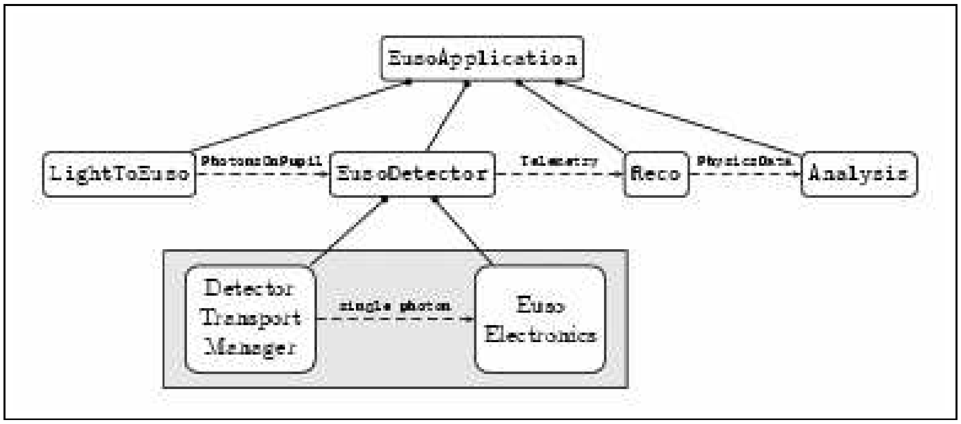

Figure 1: Basic objects in the top layers of ESAF [1].

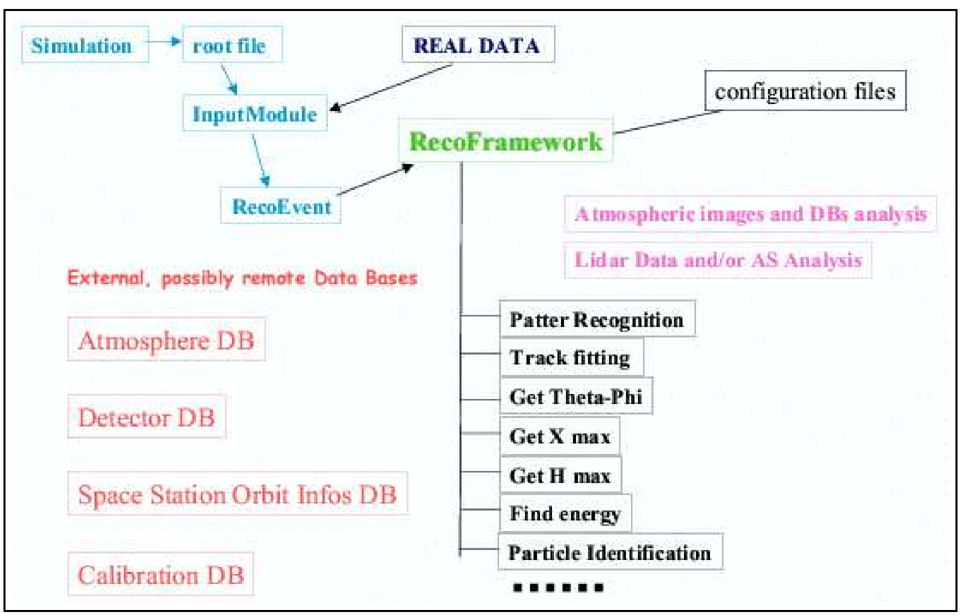

Figure 2: Proposed structure for the ESAF reconstruction module.

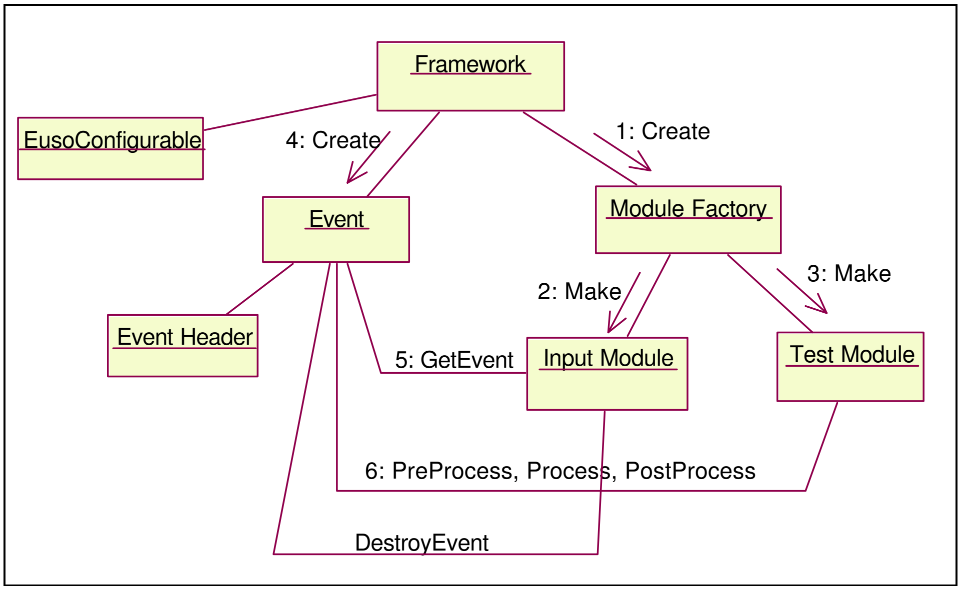

Figure 3: ESAF/Reco simplified interactions diagram (at present development stage).

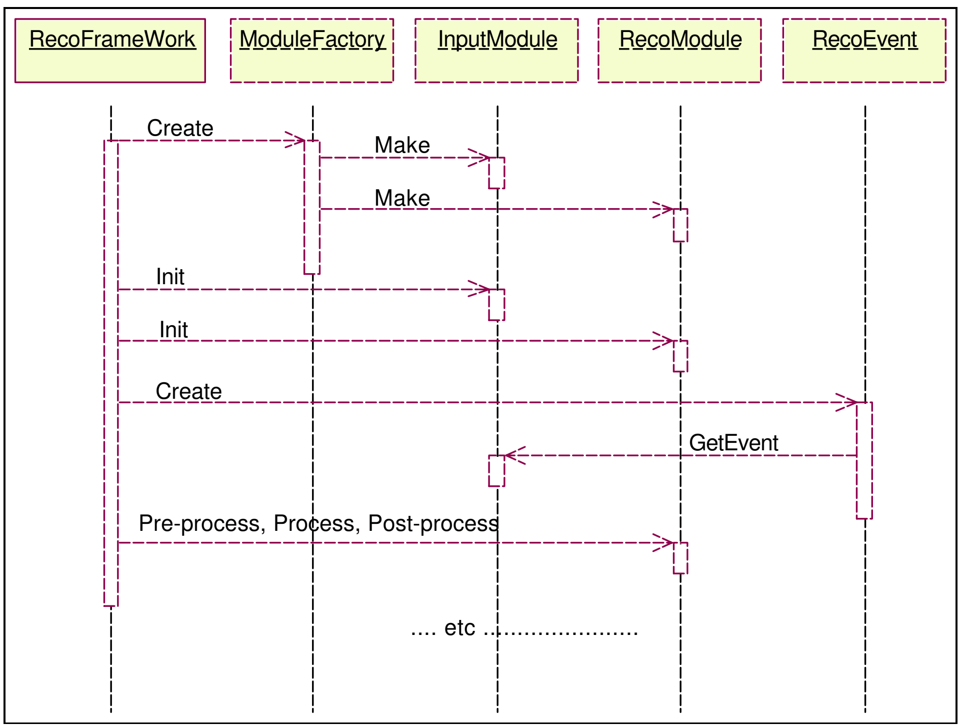

Figure 4: ESAF/Reco simplified Sequence diagram.

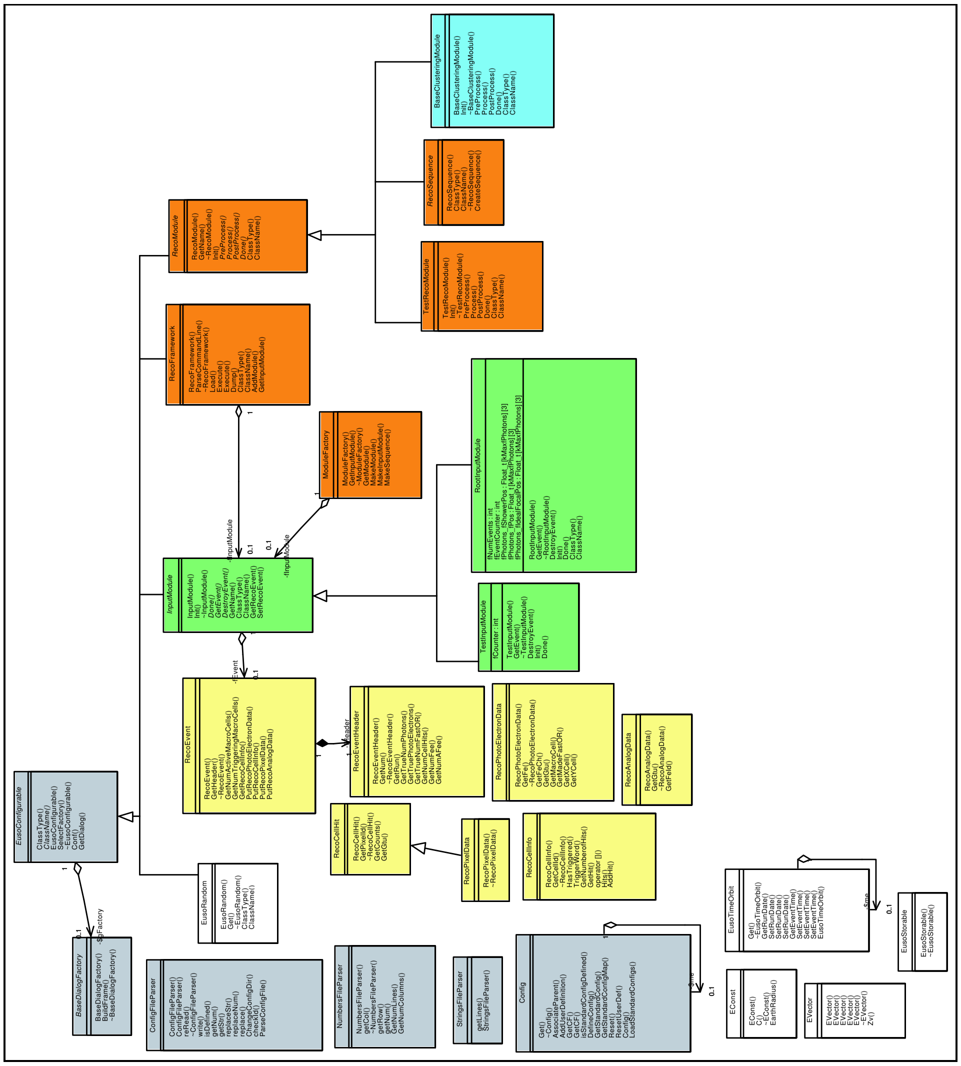

Figure 5: ESAF/Reco simplified Class diagram (at present development stage).

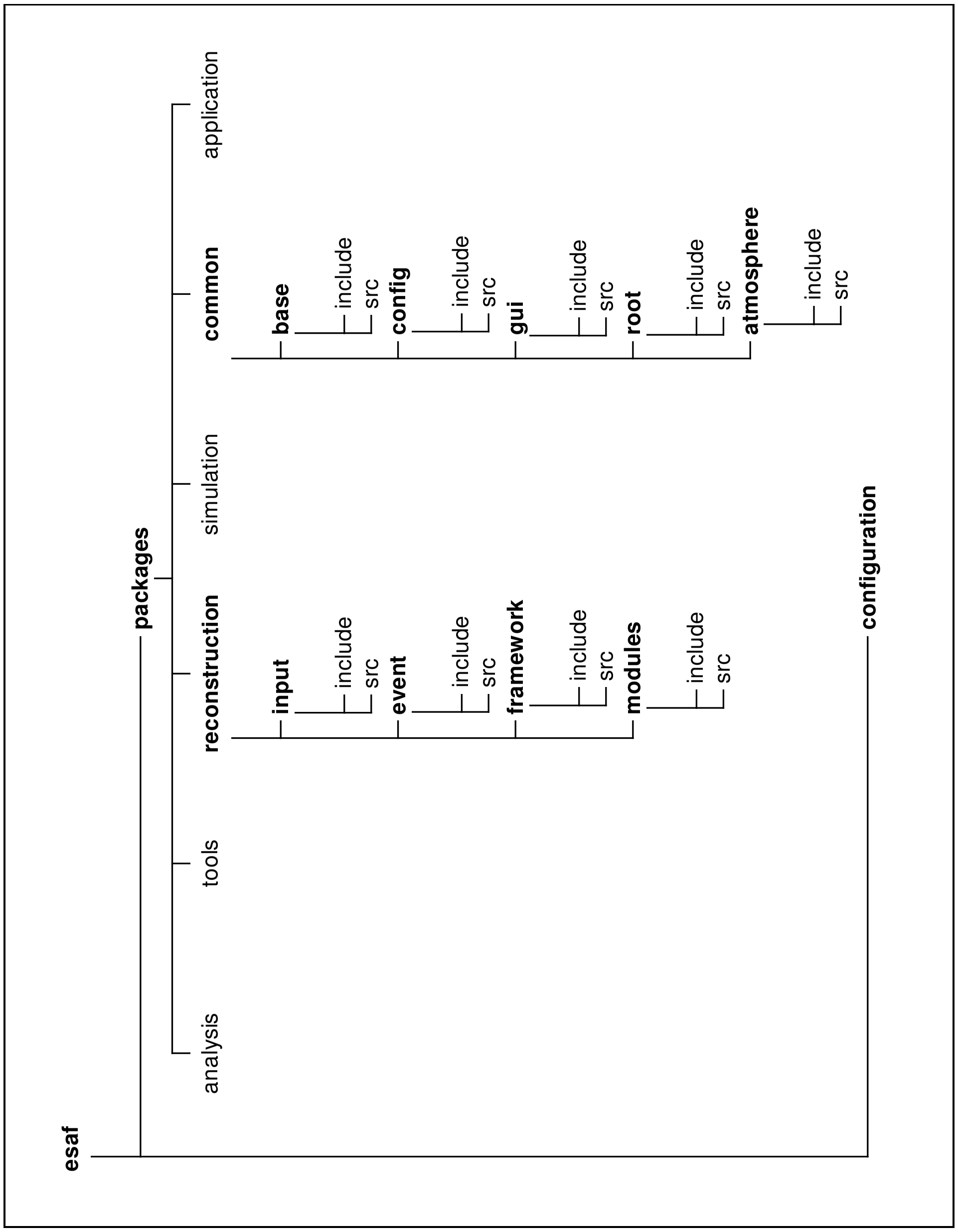

Figure 6: ESAF/Reco directory structure.